Package 'pedmut'

July 23, 2025

Title Mutation Models for Pedigree Likelihood Computations

Version 0.9.0

Description A collection of functions for modelling mutations in pedigrees with marker data, as used e.g. in likelihood computations with microsatellite data. Implemented models include equal, proportional and stepwise models, as well as random models for experimental work, and custom models allowing the user to apply any valid mutation matrix. Allele lumping is done following the lumpability criteria of Kemeny and Snell (1976), ISBN:0387901922.

License GPL-3

URL https://github.com/magnusdv/pedmut

Depends R (>= 4.2.0)

Imports lpSolve

Suggests testthat

Encoding UTF-8

Language en-GB

RoxygenNote 7.3.2

NeedsCompilation no

Author Magnus Dehli Vigeland [aut, cre] (ORCID: <https://orcid.org/0000-0002-9134-4962>), Thore Egeland [ctb] (ORCID: <https://orcid.org/0000-0002-3465-8885>)

Maintainer Magnus Dehli Vigeland <m.d.vigeland@medisin.uio.no>

Repository CRAN

Date/Publication 2025-04-29 14:30:02 UTC

Contents

`adjustRate` *Adjust the overall mutation rate of a model*

Description

Adjusts the overall mutation rate of a model by scaling the off-diagonal matrix entries.

Usage

```
adjustRate(mutmat, newrate, afreq = NULL, rate = NULL)
```

Arguments

`mutmat`	A mutation matrix with nonzero mutation overall rate.
`newrate`	The new overall mutation rate.
`afreq`	The allele frequencies. Extracted from the mutation matrix if not provided.
`rate`	The current overall mutation rate. Calculated from the input if not provided.

Details

The adjusted matrix is calculated as `a * M + (1-a) * I`, where `M` is the original matrix, `a = newrate/rate`, and `I` is the identity matrix.

The maximum allowed value of `newrate` (to avoid negative values in the adjusted matrix) is `rate/(1 - m))`, where `m` is the smallest diagonal element in the original matrix.

Value

A new mutation matrix with the adjusted rate.

See Also

`mutRate()`

Examples

```
m = mutationMatrix("equal", afreq = c(a=0.2, b=0.3, c=0.5), rate = 0.2)
m
adjustRate(m, 0.4)
```

---

findStationary *Find the stationary frequency distribution*

Description

Finds the stationary distribution of allele frequencies, if it exists, w.r.t. a given mutation matrix.

Usage

```
findStationary(mutmat)
```

Arguments

mutmat	A mutation matrix.

Value

A vector of length `ncol(mutmat)`, or NULL.

Examples

```
m1 = mutationMatrix("equal", alleles = 1:4, rate = 0.1)
findStationary(m1)

m2 = mutationMatrix("random", alleles = 1:3, seed = 123)
a = findStationary(m2)

a %*% m2 - a  # check
```

---

getParams *Get model parameters*

Description

Extract model parameters of a mutation matrix/model.

Usage

```
getParams(mut, params = NULL, format = 1, sep = "/")
```

Arguments

mut	A mutationModel() or mutationMatrix().
params	A vector contain some or all of the words "model", "rate", "range", "rate2", "seed". If NULL (default), all present parameters are included.
format	A numeric code indicating the wanted output format. See Value.
sep	A separator character used to paste male and female values. Ignored unless format = 3.

Value

When mut is a mutationModel with different male/female parameters, the output format is dictated by the format option, with the following possibilities:

1. A data frame with 2 rows labelled 'female' and 'male'.
2. A data frame with 1 row and female/male columns suffixed by .F/.M respectively.
3. A data frame with 1 row, in which female/male values are pasted together (separated with sep) if different.

If mut is a mutationMatrix the output always has 1 row.

Examples

```
M = mutationModel("equal", 1:2, rate = list(female = 0.2, male = 0.1))
getParams(M)
getParams(M, format = 2)
getParams(M, format = 3)
getParams(M, format = 3, sep = "|")
```

---

isMutationModel *Test for mutation matrix/model*

Description

Test for mutation matrix/model

Usage

```
isMutationModel(x)

isMutationMatrix(x)
```

Arguments

x	Any object.

Value

TRUE or FALSE

Examples

```
mat = mutationMatrix("equal", alleles = 1:2, rate = 0.1)
isMutationMatrix(mat)

isMutationModel(mat) # FALSE (not a complete model)

mod = mutationModel(mat)
isMutationModel(mod)
```

---

lumpedMatrix *Combine alleles in a mutation matrix*

Description

Reduce a mutation matrix by combining a set of alleles into one *lump*, if this can be done without distorting the mutation process of the remaining alleles. Such allele lumping can give dramatic efficiency improvements in likelihood computations with multi-allelic markers, in cases where only some of the alleles are observed in the pedigree.

Usage

```
lumpedMatrix(mutmat, lump, afreq = NULL, check = TRUE, labelSep = NULL)

lumpedModel(mutmod, lump, afreq = attr(mutmod, "afreq"), check = TRUE)
```

Arguments

mutmat	A mutationMatrix object, typically made with mutationMatrix().
lump	A vector containing the alleles to be lumped together, or a list of several such vectors.
afreq	A vector with allele frequencies, of the same length as the size of mutmat. Extracted from the model if not given.
check	A logical indicating if lumpability (i.e., the row-sum criterium of Kemeny & Snell) should be checked before lumping. Default: TRUE.
labelSep	A character used to name lumps by pasting allele labels. (For debugging.)
mutmod	A mutationModel object, typically made with mutationModel().

Details

The lumping implemented in this function is based on the Markov chain lumping theory by Kemeny & Snell (1976). For other, specialised lumping, see lumpMutSpecial().

Value

A reduced mutation model. If the original matrix has dimensions $n \times n$, the result will be $k \times k$, where $k = n - length(lump) + 1$.

References

Kemeny & Snell (1976). *Finite Markov Chains*. Springer.

See Also

`lumpMutSpecial()`.

Examples

```
af = c(.1, .2, .3, .4)
names(af) = 1:4

### Example 1: Lumping a mutation matrix
mat = mutationMatrix("eq", afreq = af, rate = 0.1)
mat

# Lump
lumpedMatrix(mat, lump = 3:4)
lumpedMatrix(mat, lump = 2:4)

# Example 2: Full model, proportional
mutrate = list(male = 0.1, female = 0.2)
mod = mutationModel("prop", afreq = af, rate = mutrate)
mod

# Lump
lumpedModel(mod, lump = 2:4)
```

---

lumpMutSpecial *Special lumping of mutation models*

Description

This function implements methods for special, or pedigree-aware, allele lumping. This is typically attempted if the model is not generally lumpable as determined by `alwaysLumpable()`. Note that the resulting lumped model is tailor-made for a specific likelihood calculation, and may violate the properties of a well-defined mutation model.

Usage

```
lumpMutSpecial(mut, lump, uSign, afreq = NULL, verbose = TRUE)
```

Arguments

mut	A square mutation matrix; typically a mutationMatrix() or mutationModel().
lump	A vector containing the alleles to be lumped together.
uSign	The U-signature of the pedigree for which lumping is attempted. See Details.
afreq	A vector with allele frequencies, of the same length as the size of mut. Extracted from the model if not given.
verbose	A logical.

Details

The lumping procedure depends on the location of untyped individuals in the pedigree, summarised by the so-called U-signature:

- F-depth: The length of the longest chain of untyped, starting with a founder
- F-width: The maximum number of children of an untyped founder
- N-depth: The length of the longest chain of untyped, starting with a nonfounder
- N-width: The maximum number of children of an untyped nonfounder

Value

A reduced mutation model, if lumping was possible, otherwise the original model is returned unchanged.

See Also

lumpedModel().

Examples

```
af = rep(0.05, 20)
names(af) = 1:20
m = mutationMatrix("random", afreq = af, rate = 0.1, seed = 1)

# Degree 1 lumping
mL = lumpMutSpecial(m, lump = 3:20, uSign = c(1,1,0,0))
mL

# Check
afL = attr(mL, "afreq")
stopifnot(sum(af * m[, 1]) == sum(afL * mL[, 1]))

# Degree 2
mL2 = lumpMutSpecial(m, lump = 3:20, uSign = c(1,2,0,0))
mL2
afL2 = attr(mL2, "afreq")

stopifnot(all.equal(af %*% m[, 1]^2, afL2 %*% mL2[, 1]^2),
          all.equal(af %*% m[, 2]^2, afL2 %*% mL2[, 2]^2),
          all.equal(af %*% ( m[, 1]*m[, 2]),
```

```
                    afL2 %*% (mL2[, 1]*mL2[, 2])))
```

`makeReversible` *Transformations to reversibility*

Description

This function implements three methods for transforming a mutation model (`M,p`) into a reversible one, (`R,p`). All methods are based on Metropolis- Hastings proposal functions.

Usage

```
makeReversible(
  mutmat,
  method = c("BA", "MH", "PR"),
  adjust = TRUE,
  afreq = NULL
)
```

Arguments

`mutmat`	A `mutationMatrix()` or `mutationModel()`.
`method`	A character indicating which transformation to use. Either "BA" (Barker), "MH" (Metropolis-Hastings) or "PR" (preserved rate).
`adjust`	Logical. If TRUE (default), the overall mutation rate is adjusted to preserve the original rate; see `adjustRate()`. Not relevant for method "PR", which by construction always preserves the overall rate.
`afreq`	A vector of allele frequencies. Extracted from `mutmat` if not provided.

Details

These transformations may also be applied through the `transform` argument of `mutationMatrix()` and `mutationModel()`.

Value

A reversible mutation matrix with the same allele frequencies.

Examples

```
m = mutationMatrix("equal", afreq = c(a=0.2, b=0.3, c=0.5), rate = 0.2)
makeReversible(m, "BA")
makeReversible(m, "MH")
makeReversible(m, "PR")

makeReversible(m, "BA", adjust = FALSE)  # rate differs!
```

```
# Apply to full model with different female/male rates
mod = mutationModel("equal", afreq = c(a=0.2, b=0.3, c=0.5),
                    rate = list(female = 0.1, male = 0.2))
modR = makeReversible(mod)
```

`makeStationary` *Transformation (stabilisation) to stationarity*

Description

For a given mutation model `(M,p)`, transform `M` into another mutation matrix `S` such that `S` is stationary with respect to `p`. Several methods for doing this are described by Simonsson and Mostad (2016); only the "PM" method is included here.

Usage

```
makeStationary(mutmat, afreq = NULL, method = "PM")
```

Arguments

`mutmat`	A square mutation matrix; typically a `mutationMatrix()` or `mutationModel()`.
`afreq`	A vector of allele frequencies. Extracted from `mutmat` if not provided.
`method`	A character string indicating the method to use. Currently only "PM" is implemented.

Details

These transformations may also be applied by setting `transform = "PM"` in `mutationMatrix()` or `mutationModel()`.

For details about the transformation, see Simonsson and Mostad (2016).

This function is a slightly optimised version of the `stabilize()` method in the Familias R package.

Value

An object of the same class the input `mutmat`; either a matrix, a `mutationMatrix` or a `mutationModel`.

References

Simonsson & Mostad (2016). *Stationary mutation models*. Forensic Sci. Int. Genet. 23:217–225. doi:10.1016/j.fsigen.2016.04.005

Examples

```
afreq = c(`1` = .2, `2` = .3, `3` = .5)
m = mutationMatrix("step", afreq = afreq, rate=0.1, rate2=0.01, range=0.1)
m
makeStationary(m, afreq = c(.3,.3,.4))


### Example with full model (i.e., male and female)

M = mutationModel("equal", afreq = afreq, rate = list(male=0.1, female=0.2))
M
makeStationary(M)
```

`maxRate` *Upper limits for overall mutation rate for the stepwise reversible model.*

Description

Upper limits for overall mutation rate for the stepwise reversible model.

Usage

```
maxRate(alleles, afreq, range)
```

Arguments

`alleles`	A character vector with allele labels.
`afreq`	A numeric vector of allele frequencies.
`range`	A positive number.

Value

A vector of two numbers named `UW` and `UB`. The first of these is the maximum overall mutation rate for a well-defined stepwise reversible mutation matrix with the given input. The latter (UB) is the upper limit of the overall mutation rate under the additional restraint that the model is bounded by `afreq`.

Author(s)

Thore Egeland.

`model_properties` *Mutation model properties*

Description

Functions that check various properties of a mutation model, including stationarity, reversibility and lumpability.

Usage

```
isStationary(mutmat, afreq = NULL)

isReversible(mutmat, afreq = NULL)

isBounded(mutmat, afreq = NULL)

isLumpable(mutmat, lump)

alwaysLumpable(mutmat)
```

Arguments

`mutmat`	A `mutationMatrix()` or a `mutationModel()`.
`afreq`	A vector with allele frequencies, of the same length as the size of `mutmat`.
`lump`	A character vector containing a nonempty set of allele labels.

Details

The function `isBounded()` checks that a mutation model is *bounded* by the allele frequencies, i.e., that `mutmat[i,j] <= afreq[j]` whenever `i` is not equal to `j`.

Lumpability is a property of a mutation model that allows aggregating alleles into groups, or *lumps*, without changing the overall mutation process. The functions `isLumpable()` and `alwaysLumpable()` checks lumpability using the row-sum criterion given by Kemeny & Snell (1976). Note that lumping may be possible even if the model is not generally lumpable; see `lumpMutSpecial()` for details.

For each of these functions, if `mutmat` is a `mutationModel` object, i.e., with male and female components, the output is TRUE if and only if both components satisfy the property in question.

Value

Each of these functions returns TRUE of FALSE.

References

Kemeny & Snell (1976). *Finite Markov Chains*. Springer.

Examples

```
# "proportional" models are stationary and reversible
afr = c(0.2, 0.3, 0.5)
m_prop = mutationMatrix(model = "prop", alleles = 1:3, afreq = afr, rate = 0.1)
stopifnot(isStationary(m_prop, afr), isReversible(m_prop, afr))

# "equal" model is stationary and reversible only when freqs are equal
m_eq = mutationMatrix(model = "eq", alleles = 1:3, rate = 0.1)
stopifnot(isStationary(m_eq, rep(1/3, 3)), isReversible(m_eq, rep(1/3, 3)))
stopifnot(!isStationary(m_eq, afr), !isReversible(m_eq, afr))

# "equal" and "proportional" models allow allele lumping
stopifnot(isLumpable(m_eq, lump = 1:2))
stopifnot(isLumpable(m_prop, lump = 1:2))

# In fact lumpable for any allele subset
stopifnot(alwaysLumpable(m_eq), alwaysLumpable(m_prop))
```

`mutationMatrix` *Mutation matrix*

Description

Construct mutation matrices for pedigree likelihood computations.

Usage

```
mutationMatrix(
 model = c("custom", "dawid", "equal", "proportional", "random", "onestep", "stepwise",
    "trivial"),
  matrix = NULL,
  alleles = NULL,
  afreq = NULL,
  rate = NULL,
  seed = NULL,
  rate2 = NULL,
  range = NULL,
  transform = NULL,
  validate = TRUE
)

validateMutationMatrix(mutmat, alleles = NULL)
```

Arguments

`model`	A string: either "custom", "dawid", "equal", "proportional", "random", "onestep", "stepwise" or "trivial".

matrix	When model is "custom", this must be a square matrix with nonnegative real entries and row sums equal to 1.
alleles	A character vector (or coercible to character) with allele labels. Required in all models, except "custom" if matrix has dimnames.
afreq	A numeric vector of allele frequencies. Required in model "proportional".
rate	A number between 0 and 1. Required in models "equal", "proportional", "stepwise" and "onestep".
seed	A single number. Optional parameter in the "random" model, passed on to set.seed().
rate2	A number between 0 and 1. The mutation rate between integer alleles and microvariants. Required in the "stepwise" model.
range	A positive number. The relative probability of mutating n+1 steps versus mutating n steps. Required in the "stepwise" and "dawid" models. Must be in the interval (0,1) for the "dawid" model.
transform	Either NULL (default) or one of the strings "MH", "BA", "PR", "PM". See Details.
validate	A logical (default: TRUE) indicating whether to validate custom models.
mutmat	An object of class mutationMatrix.

Details

Descriptions of the models:

- custom: Allows any mutation matrix to be provided by the user, in the matrix parameter.
- dawid: A reversible model for integer-valued markers, proposed by Dawid et al. (2002).
- equal: All mutations equally likely; probability $1 - rate$ of no mutation.
- proportional: Mutation probabilities are proportional to the target allele frequencies.
- random: This produces a matrix of random numbers, where each row is normalised so that it sums to 1. If rate (and afreq) is provided, the mutation matrix is conditional on the overall mutation rate.
- onestep: A mutation model for markers with integer alleles, allowing mutations only to the nearest neighbours in the allelic ladder. For example, 10 may mutate to either 9 or 11, unless 10 is the lowest allele, in which case 11 is the only option. This model is not applicable to loci with non-integer microvariants.
- stepwise: A common model in forensic genetics, allowing different mutation rates between integer alleles (like 9) and non-integer microvariants (like 9.3). Mutation rates also depend on step size, as controlled by the range parameter.
- trivial: The identity matrix, implying that no mutations are possible.

If transform is non-NULL, the indicated transformation is applied to the matrix before returning. Currently, there are 4 available options:

- MH, BA, PR: See makeReversible()
- PM: See makeStationary()

Value

An object of class `mutationMatrix`, essentially a square numeric matrix with various attributes. The matrix has entries in `[0, 1]` and all rows sum to 1. Both row names and column names are the allele labels.

Examples

```
mutationMatrix("equal", alleles = 1:3, rate = 0.05)

mutationMatrix("random", afreq = c(a=0.3, b=0.7), rate = 0.05, seed = 1)
```

---

mutationModel *Mutation models*

Description

Constructor for the class `mutationModel`. An object of this class is essentially a list of two mutation matrices, named "female" and "male".

Usage

```
mutationModel(
  model,
  alleles = NULL,
  afreq = NULL,
  matrix = NULL,
  rate = NULL,
  rate2 = NULL,
  range = NULL,
  seed = NULL,
  transform = NULL,
  validate = TRUE
)

validateMutationModel(mutmod, alleles = NULL)

sexEqual(mutmod)
```

Arguments

`model` Either:

- a `mutationModel` object (returned unchanged after validation)
- a single `mutationMatrix` object (will be applied to both genders)
- a list of two `mutationMatrix` objects, named "female" and "male"
- a single model name (see `mutationMatrix()` for valid options)

- a list of two model names, named "female" and "male"

alleles	A character vector with allele labels; passed on to mutationMatrix().
afreq	A numeric vector of allele frequencies; passed on to mutationMatrix().
matrix	A matrix, or a list of two matrices (named "female" and "male")
rate	A numeric mutation rate, or a list of two (named "female" and "male")
rate2	A numeric mutation rate, or a list of two (named "female" and "male"). Required in the "stepwise" model; see mutationMatrix() for details.
range	A positive number, or a list of two (named "female" and "male"). Required in the "stepwise" model; see mutationMatrix() for details.
seed	An integer, or a list of two (named "female" and "male").
transform	Either NULL (default) or one of the strings "MH", "BA", "PR", "PM". See mutationMatrix().
validate	A logical, by default TRUE.
mutmod	A mutationModel object.

Value

An object of class mutationModel. This is a list of two mutationMatrix objects, named "female" and "male", and the following attributes:

- sexEqual: TRUE if both genders have identical models.
- alwaysLumpable: TRUE if both genders have models that are lumpable for any allele subset.

Examples

```
# "Equal" model, same parameters for both genders
M1 = mutationModel("eq", alleles = 1:2, rate = 0.1)
M1

# Different mutation rates
M2 = mutationModel("eq", alleles = 1:2, rate = list(male = 0.1, female = 0.01))
M2

stopifnot(identical(M1$male, M1$female), identical(M2$male, M1$male))

# A custom mutation matrix:
mat = matrix(c(0,0,1,1), ncol = 2, dimnames = list(1:2, 1:2))
M3 = mutationModel(model = "custom", matrix = mat)

# Under the hood arguments are passed to `mutationMatrix()`.
# Alternatively, this can be done explicitly in the `model` argument
M4 = mutationModel(model = mutationMatrix("custom", matrix = mat))

stopifnot(identical(M3, M4))

# The latter strategy is needed e.g. in `pedtools::marker()`, which gives the
# user access to `model`, but not `matrix`.
```

mutRate *Overall mutation rate*

Description

Calculate the overall mutation rate at a locus, given a mutation model an a set of allele frequencies.

Usage

```
mutRate(mutmat, afreq = NULL)
```

Arguments

mutmat	A mutationMatrix() or mutationModel().
afreq	A vector of allele frequencies.

Details

The mutation rate is found by the formula `1 - sum(diag(mutmat) * afreq)`.

If `mutmat` is a full `mutationModel()`, the rate is calculated separately for the male and female matrices.

Value

A single number, or (if `mutmat` is a `mutationModel()` and the female and male rates differ) a list of two numbers, named "female" and "male".

Examples

```
m = mutationMatrix("stepwise", alleles = 1:4, afreq = c(.1,.2,.3,.4),
                   rate = 0.01, rate2 = 1e-6, range = 0.1)
r = mutRate(m)

stopifnot(all.equal(r, 0.01))
```

stabilize *Stabilization of mutation matrix*

Description

NB: REPLACED BY makeStationary. Produces a mutation matrix close to the input `mutmat`, for which the given frequency vector is the stationary distribution. Several methods for doing this are described by Simonsson and Mostad (2016); only the "PM" method is included here.

Usage

```
stabilize(mutmat, afreq = NULL, method = "PM", details = FALSE)
```

Arguments

mutmat	A mutation matrix.
afreq	A vector of allele frequencies.
method	Either "DP", "RM" or "PM". Currently only "PM" is implemented.
details	A logical. If TRUE, the complete Familias output is included.

Details

This function is based on, and reuses code from, the `stabilize()` method of the Familias R package.

Value

An object of the same class the input `mutmat`; either a matrix, a `mutationMatrix` or a `mutationModel`.

Author(s)

Petter Mostad, Thore Egeland, Ivar Simonsson, Magnus D. Vigeland

References

Simonsson, Mostad: Stationary Mutation models. (FSI: Genetics, 2016).

Examples

```
afreq = c(`1` = .2, `2` = .3, `3` = .5)
m = mutationMatrix("stepwise", afreq = afreq,
                   rate = 0.1, rate2 = 0.01, range = 0.1)
m
stabilize(m)


### Example with full model (i.e., male and female)

M = mutationModel("stepwise", alleles = 1:3, afreq = afreq,
                   rate = list(male = 0.1, female = 0.2),
                   rate2 = 0.01, range = 0.1)
M
stabilize(M)
```

`stepwiseReversible` *Dawid's reversible stepwise model*

Description

#' A reversible stepwise mutation model is created following the approach of Dawid et al. (2002).

Usage

```
stepwiseReversible(alleles, afreq, rate, range, maxRateOnly = FALSE)
```

Arguments

`alleles`	A vector of integer integers.
`afreq`	A numeric vector of allele frequencies.
`rate`	A numeric mutation rate.
`range`	A positive number.
`maxRateOnly`	A logical, by default FALSE. See Value.

Details

NB: This function is deprecated: Use `mutationMatrix(model = "dawid", ...)` **instead.**

For the stepwise reversible model, the mutation rate $r_{i,j}$, $i \neq j$ is proportional to the overall mutation rate λ for given values of the range, the allele frequency p_i and n, the number of alleles. Hence, one can determine bounds UW and UB so that the model is well defined if $\lambda \leq UW$ and bounded, i.e., $r_{i,j} \leq p_j$, $i \neq j$, if $\lambda \leq UB$, The bounds UW and UB are computed.

Value

A reversible stepwise mutation model with overall mutation rate equal to `rate`.

If `maxRateOnly` is TRUE, the function returns a vector of two numbers named `UW` and `UB`. The first of these is the maximum overall mutation rate for a well-defined stepwise reversible mutation matrix with the given input. The latter (UB) is the maximum rate under the additional restraint that the model is bounded by `afreq`.

Author(s)

Thore Egeland.

Examples

```
stepwiseReversible(alleles = 1:3,
                   afreq = c(0.2, 0.3,  0.5),
                   rate = 0.001,
                   range = 0.1)
```

```
stepwiseReversible(alleles = 1:3,
                   afreq = c(0.2, 0.3,  0.5),
                   range = 0.1,
                   maxRateOnly = TRUE)

# Model not well defined:
## Not run:
stepwiseReversible(alleles = 1:3,
                   afreq = c(0.2, 0.3,  0.5),
                   rate = 0.7,
                   range = 0.1)

## End(Not run)
```

Index